Finishing *que data structures

2/12/16
Announcements

• HW due Tuesday: Sorting and GUI layout

• Reading for Monday: Chapter 12 and Java Interlude 5 (Lists and Iterators)

• Quiz today:
  – Logs
  – Sorting
  – Queue, deque, priority queue
Lab: GUI design

• Components
  – JFrame, JButton, JPanel, JLabel

• Listeners & event-driven programming

• Layout managers
  – FlowLayout, GridLayout, BorderLayout
Summary

• Queue ADT:
  – enqueue adds an element at the back
  – dequeue removes one from the front

• Array-based queue implementations
  – put the front in cell 0: bad for dequeue
  – put the back in cell 0: bad for enqueue
ArrayQueue: another option

• Neither of those solutions is very good as they both involve moving all the existing data elements, which has high time cost

• Idea: Instead of moving data elements to a fixed position for front when removing, let front advance through the array

Hmm....what do we do when we now add an element to that queue at the rear? What happens when we remove several elements, and front catches up with rear...
ArrayQueue: Using a *circular* array underlying data structure

**Solution**: Be more creative!

View the array as *circular* and allow both *front* and *rear* to advance through (around) the array.

This will require *no* data movement for enqueues or dequeues!
public E dequeue(){
    // potential issue if empty,
    // for now, assume not empty
    size--;
    E e = array[front];
    <YOUR CODE HERE>
    return e;
}

Select the correct code to insert from below:

A  front++;  
   if(front == array.length)  
     front = 0;

B  rear = rear-1;  
   if(rear <0)  
     rear = array.length-1;

C  for(int i= 0; i<rear; i++) {
    array[i] = array[i+1];  
  }
   rear = rear -1;  
   if(rear <0)  
     rear = array.length-1;

D  Not exactly one of the above
public E dequeue(){
    // potential issue if empty,
    // for now, assume not empty
    size--;
    E e = array[front];
    <YOUR CODE HERE>
    return e;
}

Select the correct code to insert from below:

A  front++;  
    if(front == array.length)  
        front = 0;

B  rear = rear-1;  
    if(rear <0)  
        rear = array.length-1;

C  for(int i= 0; i<rare; i++) {
        array[i] = array[i+1];
    }  
    rear = rear -1;  
    if(rear < 0)  
        rear = array.length-1;

D  Not exactly one of the above
public void enqueue(E e){
    // potential issue if full,
    // for now, assume not full
    size++;
}

Select the correct code to insert from below:

A  rear++;
    if(rear == array.length)
        rear = 0;
    array[rear] = e;

B  rear++
    array[rear] = e;

C  for(int i = front; i < rear; i++) {
        array[i] = array[i+1];
    }
    array[rear] = e;
    front--;

D  None of these are correct
public void enqueue(E e) {
    // potential issue if full,
    // for now, assume not full
    // YOUR CODE HERE
    size++;
}

Select the correct code to insert from below:

A  rear++;  
   if(rear == array.length)  
     rear = 0;  
   array[rear] = e;

B  rear++  
    array[rear] = e;

C  for(int i= front; i<rear; i++) {
    array[i] = array[i+1];
  }
  array[rear] = e;  
  front--;  

D  None of these are correct
Suppose we have a Queue implemented with a circular array. The capacity is 10 and the size is 5. Which are *not* legal values for front and rear?

<table>
<thead>
<tr>
<th></th>
<th>front</th>
<th>rear</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>C</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>E</td>
<td>Not exactly one of the above</td>
<td></td>
</tr>
</tbody>
</table>
Suppose we have a Queue implemented with a circular array. The capacity is 10 and the size is 5. Which are **not** legal values for front and rear?

<table>
<thead>
<tr>
<th></th>
<th>front</th>
<th>rear</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>C</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>E</td>
<td>Not exactly one of the above</td>
<td></td>
</tr>
</tbody>
</table>
Array-based queue

- Possible using a circular array
- A bit complicated by need to wrap indices
  - Modular arithmetic:
    Increment: \(i = (i + 1) \mod \text{vals.length};\)
    Decrement: \(i = (i - 1 + \text{vals.length}) \mod \text{vals.length};\)
- Easier with linked lists because you can make local changes